「オセロ AI」の開発

佐藤優介
A-1

1．はじめに

私たちのグループはオセロの AI と対戦するオセロゲーム「オセロ AI」を作成した。

このゲームの開発を決めた背景としては、近年のボードゲームにおける AI の台頭がある。AI の歴史としては、まず 1997 年にチェスの世界王者を倒した。2012 年から 2015 年にかけて行われた将棋の将棋電王戦では AI が勝つ試合が多かった。そこで自分も AI を開発してみたいと考え、ゲームの AI としては最も基本的であるオセロの AI を開発し、対戦することができるゲームの開発を決定した。

2．問題設定

「オセロ AI」はオセロで AI と対戦するゲームである。したがって最低限の持つべき機能として以下の機能が挙げられる。

- オセロの盤面（枠や石）を表示する。
- オセロのルールに従って石を設置、反転する。
- AI から石の設置場所を取得する。
- プレイヤーから石の設置場所を取得する。
- 試合終了時に結果を表示する。

これらの機能のみで構成されるゲームは非常に簡素なものとなることが想定される。そこで、まずはこれらの最低限の機能の実装を目指し、実装が完了した後にゲームとしての完成度を上げる機能の実装を行った。完成度を上げる機能としては以下の機能を実装した。

- AI の強さの設定機能
- 先攻後攻の設定機能
- BGM の有無と音量の調整機能

今回、開発を決定したオセロ AI のゲームは PvP 対戦型のオセロゲームに、オセロ AI の機能を追加した形となっており開発規模が大きい。またプログラミング初心者が多かったため、分野ごとに班分けし、開発を行った。具体的には、ゲームのフレームワークやユーザーインターフェースを担当する UI 班、オセロのルールに則って、オセロの盤面を管理する盤面管理クラスを担当する盤面管理班、オセロの AI クラスの開発を担当する AI 班である。

私は AI 班で AI クラスの開発を行った。詳しくは「4. 課題を解決するための主な工夫」で述べるが、AI は主に「評価」と「探索」で構成される。AI の手法としてよく挙げられる強化学習や深層学習は、オセロ AI という分野においては、盤面をもとに「評価」を行う評価関数のパラメータを決定する際に用いられる手法を指す。一方で「探索」とは自分と相手が交互に石を置いていき、到達しうる局面を先読みしていくことを指す。この「探索」と「評価」によって自分が石を置くことができる場所に点数をつけ、その点数が高い場所を強い手として石を置くことでオセロの AI を構成する。

AI クラス開発の最低限の機能としては以下のものが挙げられる。

- 石を置ける場所の評価値の比較機能
- 評価関数の設計
- 探索の実装
- 探索をより深くするため、これらの高速化

また探索の実装に際して、局面の先読みを行うには設置した場合にどのような盤面になるか調べる必要がある。そのため以下の機能が必要である。

- 設置可能場所の取得機能
- 設置後盤面の取得機能

AI の強さを向上させる挑戦としてはビットボードによる実装、ビットボードのビット演算を活かした高速化、評価関数の精度の評価、探索の効率化などがある。

3．提案するシステム構成

私たちの班はこのゲーム開発するにあたり、プログラミング言語として C++を採用した。C++は授業でも扱った言語であり、開発メンバーにプログラミング初心者が多いこともあり、この言語を採用した。また、このゲームの UI を実装するためのフレームワークとして、C++の外部ライブラリである OpenSiv3D を採用した。OpenSiv3D は 2D/3D ゲーム、メディアアート、シミュレータなど、可視化やインタラクションに関わるクリエイターのための C++ライブラリであり、短い、簡単なコードで実装することができる。

次に、図 1 にオセロ AI のシステム構成を示す。

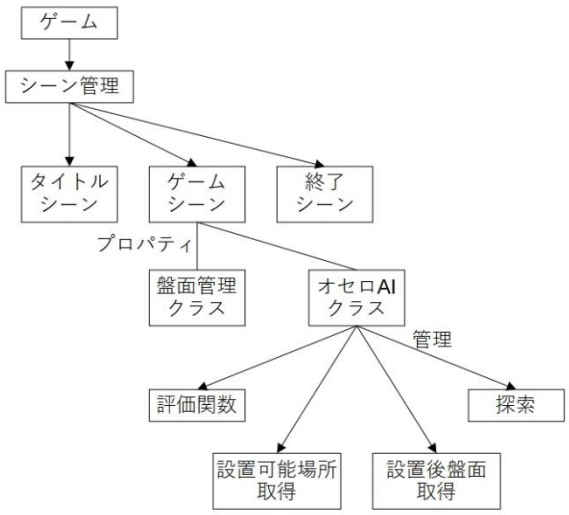

図 1　システム構成

図 1 のシーン管理、タイトルシーン、ゲームシーン、終了シーンは UI 班、盤面管理クラスは盤面管理班、オセロ AI クラスは私が属する AI 班が開発を行った。

シーン管理と各シーンは手続き型プログラミングによって実装している。ゲームシーンは盤面管理クラスとオセロ AI クラスをプロパティとして持つ。盤面管理クラスはカプセル化を行い、石の設置や盤面の情報の取得などのメンバ関数を用意することで、オセロのルールに違反する設置を防ぎ、盤面の情報を保護している。オセロ AI クラスが提供する機能としては AI のパラメータの設定と盤面の情報から AI の設置場所を返す機能である。クラスのカプセル化により AI のパラメータを保持することで、異なるパラメータを持つ AI を作成可能にし、異なるパラメータをもつ AI との対戦や AI 同士の対戦によるパラメータの評価を行えるようにした。

図 2 に AI 班が担当したオセロ AI クラスのクラス図を示す。

オセロ AI クラス

- プロパティ
  - 配置良さ評価パラメータ
  - AI 強さモード
  - 探索制限時間
  - その他の探索効率化に使用する変数
- メソッド
  - コンストラクタ
  - AI のパラメータの再設定関数
  - AI 設置場所取得関数
  - 各種データ構造の変換関数
  - 設置可能場所取得関数
  - 設置後盤面取得関数
  - ビットカウント関数
  - 評価関数
  - 再帰関数による $\alpha\beta$ 探索関数

図 2　オセロ AI クラスのクラス図

AI 班では、班員でオセロ AI クラスの仕様を決定し、メンバ関数の開発を分担して行った。私が担当した部分は以下である。

- コンストラクタ
- AI のパラメータの再設定関数
- AI 設置場所取得関数
- 評価関数
- 再帰関数による $\alpha\beta$ 探索関数

なお、評価関数の実装は私が行ったが、どのようなパラメータをもとに評価を行うかなどの設計は AI 班全体で行った。

4．課題を解決するための主な工夫

AI は「評価」と「探索」で構成されることから、AI の強さは「評価」で有利・不利をどれだけ正確に数値化できるか、「探索」でどれだけ先の局面まで読むことができるかによって決まる。探索深さを深めるためには、プログラム全体を高速化する必要がある。そこで採用したのがビットボードという手法である。ビットボードとは 8×8 マスの自分と相手の石をそれぞれ 64bit 変数に格納する手法である。これにより 64bit 変数 2 つのみで盤面を表現することができる。図 3 にビットボードによる盤面の表現を示す。

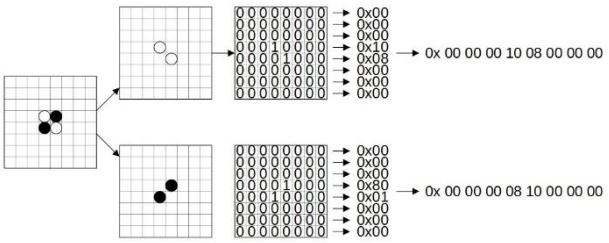

図 3　ビットボードによる盤面の表現

この手法における利点は、設置可能場所の取得と設置後盤面の作成に for 文や if 文をほとんど使わず、ビット演算のみで実行できることにある。この二つの関数は私の開発範囲ではないため詳しい説明は省略するが、例えば自分の石を white、相手の石を black、自分の着手位置を put、相手の反転する石を rev とすると、設置の処理を以下の 2 行の簡単なビット演算で行うことができる。

white ^= put | rev

black ^= rev

for 文による繰り返しや if 文による条件分岐がなく、ビット演算で高速な処理が可能となるため、今回のオセロ AI クラスに採用した。

次に「評価」を行う評価関数について述べる。今回、盤面の評価を行うパラメータとして以下の 3 つを用いた。

- 石の配置の良さ
- 石の数
- 設置可能場所の数

石の数はゲームの勝敗を決定する重要なパラメータである。設置可能場所の数は相手の手を制限し、たくさんある自分の手から有利な手を打つことができる。

石の数はビットが 1 である数を数えるビットカウント関数によって得られる。また同様に、設置可能場所の数も設置可能場所のビットボードからビットカウント関数によって得られる。これらを用いて、評価値を以下の式によって計算した。

$$\text{評価値} = \frac{\text{自分の数} - \text{相手の数}}{\text{自分の数} + \text{相手の数}}$$

また、石の数について、自分の石がゼロになると負けてしまうためこの場合は評価値を$-\infty$とし、逆に相手の石がゼロになる場合は∞とした。

次に石の配置の良さについて述べる。図 4 に配置の良さの評価例を示す。

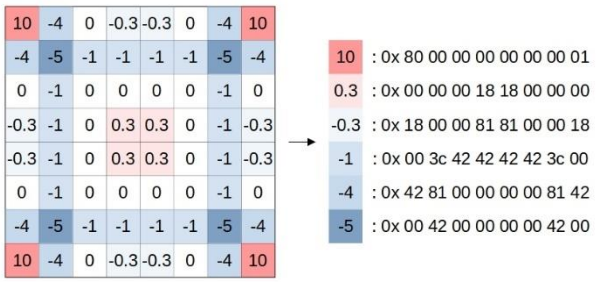

図 4　配置の良さの評価の例

図 4 の左のように取られることがない角の点数は高く、逆に角の周りは設置すると角を取られる危険性があるため点数が低くなっている。自分の石であればこの点数×1、相手の石であれば点数×−1をし、その総和を取ることで評価値とした。また、ビットボードによる高速化を活かすために、図 4 の右のように点数ごとの場所をビットボードに記録している。石のビットボードと論理積を取り、ビットカウントによりその点数の位置にある石の数を取得、最後に点数を掛けて総和を取ることで評価値を得ることができる。2 重の for 文や if 文、配列の要素への参照を用いず、高速に評価値を得ることができるよう工夫した結果このような実装となった。

これら 3 つの評価項目はゲームの進行度によって重要度が大きく変化する。序盤から中盤にかけては配置の良さが、中盤では設置可能場所の数が、終盤では勝敗に関わる石の数が重要であると考え、各評価値の重み付けとして図 5 のような関数を用いた。

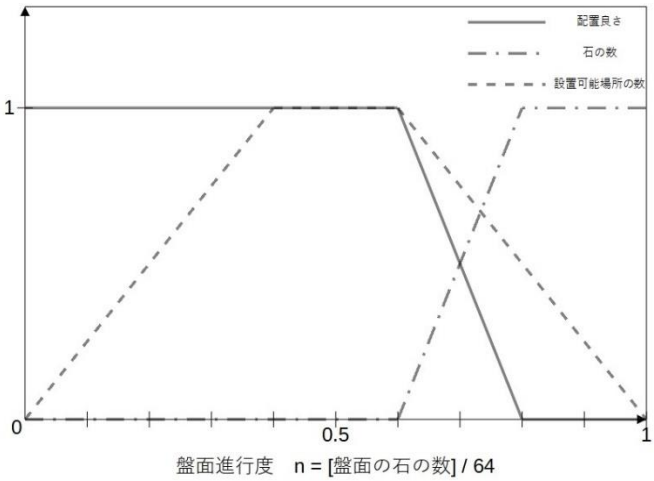

図５　評価値の重み付け関数

この関数を各評価値に掛けることで、ゲーム進行度における評価値の重要度を反映した。

このようにして得られた評価値を－1 倍することで AI を弱い AI とすることも可能となる。

次に「探索」について述べる。今回の「探索」の基礎となる考え方である MinMax 法について説明する。図 6 に MinMax 法の図を示す。

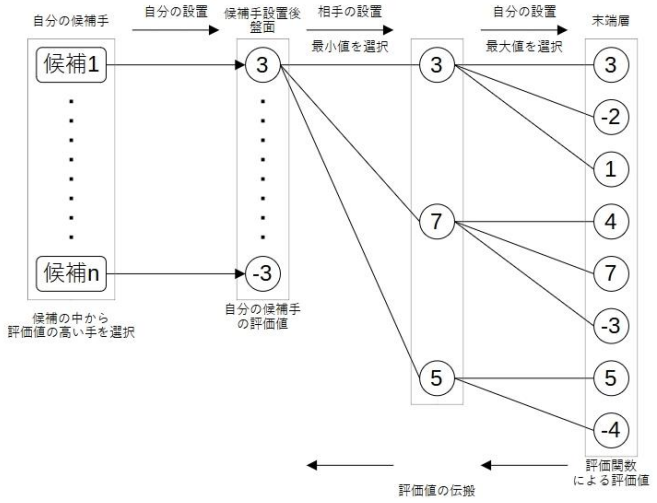

図６　MinMax 法

MinMax 法の基本的な考え方としては、「こちらにとって最悪な手を相手に打たとしても、それでも評価が高い手を打つ」である。この考え方は相手も自分も最善の手を打つことを担保として成り立っている。まず末端のノードの盤面を評価し、自分は最善の手を打つとして評価値が最大の手を選択する。逆に相手は自分にとって最悪(相手にとって最善)の手を打つと仮定して評価値が最低の手を選択する。その結果、評価値が最大となった候補手が強い手であるとして AI がこの手を選択する。MinMax 法の実装では、最大値を選択する Max 関数と最小値を選択する Min 関数が相互に互いの関数を呼び合う相互再帰関数で実装することができる。しかし、評価値の伝搬時に評価値を－1 倍することで単一の再帰関数で実装することができる。これを negaMax 法という。図 7 に negaMax 法の図を示す。

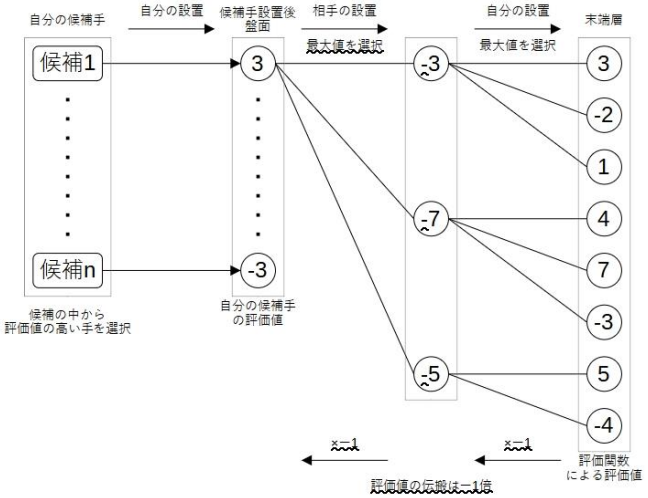

図７　negaMax 法

MinMax 法との相違点は下部波線で示している。評価値が－1 倍される部分で選択するのが最小値から最大値に変わっていることで、MinMax 法と同じ結果が得られている。

次にこの「探索」を効率化する nega$\alpha\beta$法について説明する。nega$\alpha\beta$法は$\alpha\beta$法を、MinMax 法から negaMax 法に改良したのと同様の改良をしたものである。探索を効率化するためには探索するノードの数を減らす必要がある。そこで探索する必要がないノードの探索を打ち切る「枝刈り」を行う。図 8 に nega$\alpha\beta$法の図を示す。

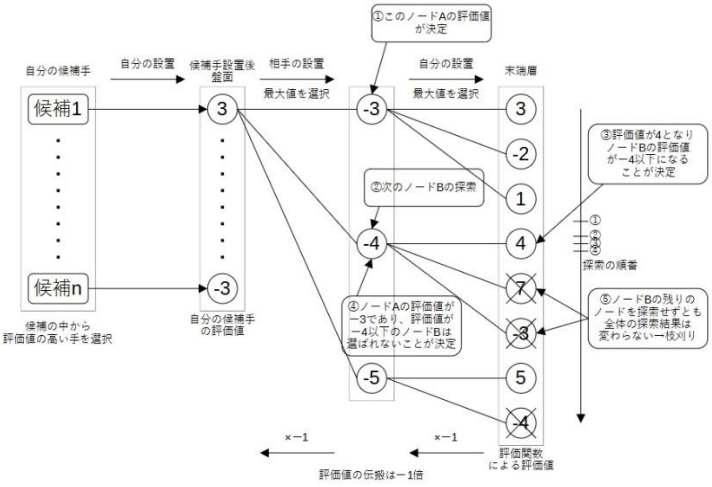

図８　nega$\alpha\beta$法

negaMax 法との違いは枝刈りにある。まず①でノード A の評価値が決定。②で次のノード B の探索を行う。ここで③において評価値が 4 となり、最大値を取り－1 倍する negaMax 法の特徴からノード B の評価値が－4 以下となることが決定する。しかし④のようにノード A の評価値は－3 であり、negaMax 法では最大値を取るためノード B が選択されないことが決定する。そのため⑤ノード B の残りのノードを探索しなくても全体の探索結果は変わらないため、ノード B の残りの探索を打ち切る。これを「枝刈り」という。枝刈りの効果は非常に大きく、探索深さを 1.5～2 倍程度深くすることができた。

ここまで紹介してきた探索手法はすべて深さ優先探索という探索である。まず、ある深さまで潜り、そのノードから親のノードに遡るように探索していく。そのため、探索深さの指定を誤るとノードの数が多くなり、探索に膨大な時間がかかることがある。そこで用いたのが反復深化深さ優先探索である。この探索手法は探索に制限時間を設け、深さを 1 から順に増やした深さ優先探索を行い、制限時間までに終了した深い探索を探索結果として適用する探索手法である。そこで nega α β 探索関数に制限時間により探索を終了する機能を追加し、反復深化深さ優先探索を実装した。

ここで枝刈りについて効率化することができる。枝刈りは最終的に選ばれる、評価値が最も大きいノードの探索を最初に行うことで枝刈りの数を増やすことができる。そこで反復深化深さ優先探索の一つ前の深さの探索経路を保存しておき、次の深さの探索時にまずこの経路の探索を行うことで、最初に評価値が高いノードの探索を期待できる。この探索経路はメンバ変数に保存しておくことで、再帰関数から探索経路の参照を行えるよう実装した。

図 9 に枝刈りの効率化の有無による探索深さと探索局面数の関係の測定結果とその近似曲線を示す。測定は探索時間 10 秒の AI 同士で対戦させた際の、反復深化深さ優先探索で最後に探索し終えた探索の深さとその時の探索局面数を測定した。

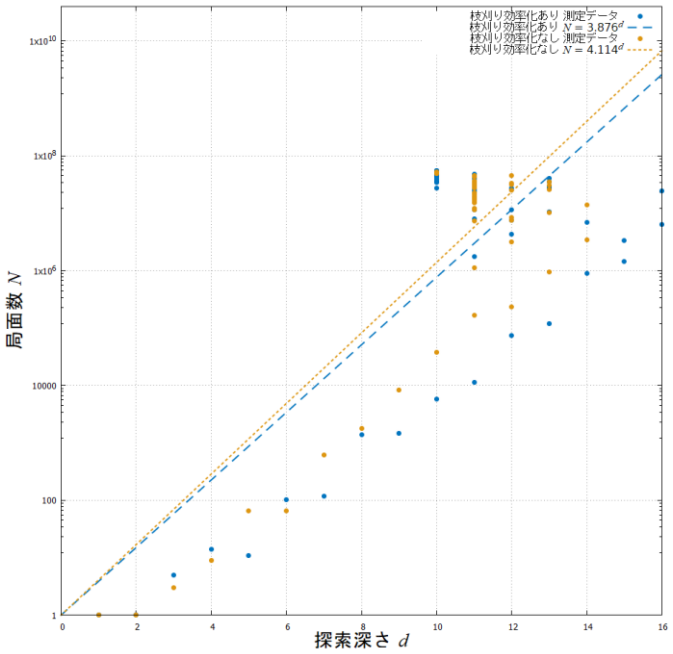

図 9　探索深さと局面数の関係

設置可能場所が各盤面で b あるとすると探索深さ d での局面数 N は $N = b^d$ となる。このことから測定データから最小二乗法により近似曲線を得ると以下のようになった。

枝刈り効率化あり：$N = 3.876^d$

$$d = 10 \rightarrow N = 0.77 \cdot 10^6$$

枝刈り効率化なし：$N = 4.114^d$

$$d = 10 \rightarrow N = 1.39 \cdot 10^6$$

枝刈り効率化ありではなしに比べて探索深さあたりの局面数が少なくなっていることが分かる。また、局面数の減少は探索深さが深くなるとより顕著に表れる。これは枝刈りの効率化によって枝刈りの数を増やせているためであると考えられる。また、図 9 を見ると枝刈り効率化ありの方が探索深さ 15 や 16 など、効率化なしに比べてより深くまで探索できていることが分かる。これは探索局面が減少したことにより、制限時間内により深くまで探索できたためであると考えられる。

このように枝刈りの効率化により探索を効率化することができた。

今回のオセロ AI ゲームでは探索制限時間を 2 秒として実装している。cpu の性能などに依存するが、2 秒間での探索局面数は約 150 万局面ほどであった。

5．プログラム実行結果

プログラムを実行するとまず図１０の画面が表示される。

図１０　開始画面

開始画面では AI の難易度を設定する。題名の下にある黄緑色の文字は時間経過で七色に変化している。次に図１１に難易度選択後の画面を示す。

図１１　先攻後攻選択画面

この画面ではプレイヤーの先攻後攻を決定する。次に図１２に先攻後攻決定後の画面を示す。

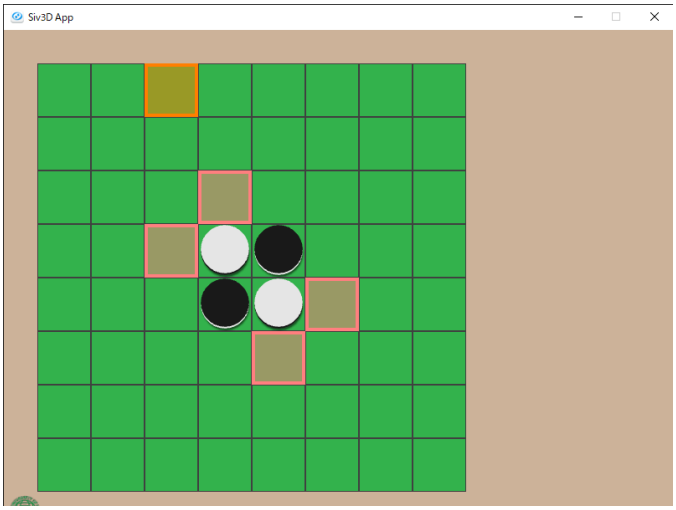

図１２　ゲームプレイ画面

画像では映っていないがマウスカーソルがあるマスがオレンジ色にハイライトされ、プレイヤーの設置可能場所はピンク色にハイライトされる。次に図１３に左下のロゴをクリックすると表示される画面を示す。

図１３　設定画面

この画面は設定を変更する画面で設定項目をクリックするとその設定項目の設定変更画面へ移動する。また､戻るボタンを押すとゲームプレイ画面へ戻る。図１４に図１３の BGM ボタンをクリックすると表示される画面を示す。

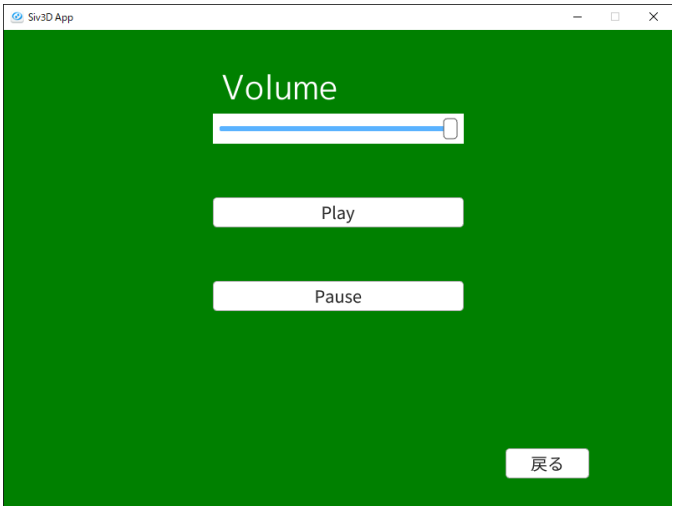

図１４　BGM 設定画面

この画面では BGM の設定を変更できる。初期では BGM は無効であるが、Play ボタンを押すことで有効化できる。Pause ボタンを押すと無効化される。また Volume のスライダーを動かすことで BGM の音量を調整できる。図１５にオセロの終了条件が満たされた際に表示される画面を示す。

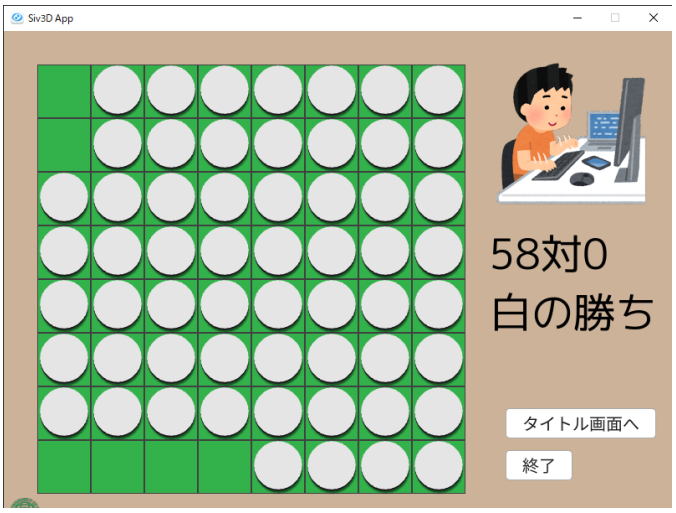

図１５　終了画面

オセロの終了条件は「どちらも設置できなくなる」である。終了画面では、お互いの石の数とどちらの勝利かが表示される。またタイトル画面へのボタンを押すと図９の開始画面へ戻る。終了ボタンを押すとこのプログラムが終了する。なお、このテストプレイ時は真面目にプレイしたが AI に全滅させられてしまった。

６．主張点の明確化

私たちはオセロの AI を開発し、AI と対戦できるゲーム「オセロ AI」を開発した。

オセロゲームは類似するような対戦型のオセロゲームが数多く存在するが、PvP の対戦型オセロゲームに比べて、AI と対戦するオセロゲームはオセロのゲーム部分の製作に加え、AI の開発が必要となる。AI の開発手法は数多く存在するが、AI が打つ手の特徴は開発者のチューニングに大きく左右される。今回、開発した AI のコンセプトは、相手の打てる場所を制限し、探索局面を減少させることで深くまで探索を行い、相手に成す術を与えずに勝つというものである。この特徴はオセロ初心者との対戦で顕著に表れる。実際に開発メンバーや知り合いと対戦してもらうと、中盤から終盤にかけて打てる場所が 2 か所や 1 か所、または打てる場所がなくスキップとなることが多く、そのまま AI が勝利していた。このように特徴を持った AI を意図して開発できたことは我々のプロジェクトの新規性であると考える。

７．おわりに

以上が我々の作成したオセロゲーム「オセロ AI」である。

このプロジェクトには多くの改善点がある。グラフィック部分であれば石を設置するモーションや石がひっくり返るモーションの追加、ゲーム性であればプロパティファイルを用意しゲームの設定や AI の設定を記述する、AI 部分であれば AI 同士を対戦させる深層学習によって AI を強化させるなど、様々な案がある。しかしこれらの実装難易度は非常に高く、今回の実装は見送った。

参考文献

[1] OpenSiv3D，チュートリアル，https://siv3d.github.io/ja-jp/tutorial/basic/，(2021 年 12 月)

[2] オセロをビットボードで実装する https://qiita.com/sensuikan1973/items/459b3e11d91f3cb37e43，(2021 年 11 月)

[3] AI 全般，http://hitsujiai.blog48.fc2.com/blog-category-4.html，(2021 年 10 月)